

In my Wish List, an Automated Tool for Fail-Secure Design Analysis: an Alloy-Based Feasibility Draft

Gurvan Le Guernic

DGA Maîtrise de l'Information
35998 Rennes Cedex 9, France

Gurvan.Le-Guernic@intradef.gouv.fr

A system is said to be fail-secure, sometimes confused with fail-safe, if it maintains its security requirements even in the event of some faults. Fail-secure analyses are required by some validation schemes, such as some Common Criteria or NATO certifications. However, it is an aspect of security which has been overlooked by the community. This paper attempts to shed some light on the fail-secure field of study by: giving a definition of fail-secure as used in those certification schemes, and emphasizing the differences with fail-safe; and exhibiting a first feasibility draft of a fail-secure design analysis tool based on the Alloy model checker.

1 Introduction

Security is an information system property of continuously growing importance. It is widely studied by academics, taken into account by industries in the design and implementation of their systems and equipments, and a concern for customers. However, it is a property which encompasses some subproperties which seem to attract less attention from the academic community and general purpose industries, but are of high interest for some specific industries. *Fail-secure* behavior is such a subproperty.

Fail-secure, which is sometimes confused with the concept of fail-safe, is the property of a system or equipment to maintain its *security* properties even in the event of *some* faults. Section 2 defines this concept for the purpose of this paper and some evaluation schemes, such as the Common Criteria (CC) [1] or North Atlantic Treaty Organization (NATO) Military Committee (NAMILCOM) certifications [9]. In addition to be required for some of those evaluations, fail-secure design analyses are welcome in the early stages of the engineering of fail-secure systems. Indeed, making a system fail-secure imposes additional constraints on the functional and physical architectures of the system. At the certification stage, it is often difficult to modify a product in order to take into account those constraints, particularly concerning its physical architecture, which advocates for the need of early fail-secure design analyses.

This aspect of security, fail-secure behavior, has not attracted much interest from the academic community. Therefore, there are very few tools and techniques for fail-secure analysis of software or hardware architectures (Sect. 2.1). Nowadays, real world fail-secure software or hardware architectures analyses are mostly carried out manually. Hence, such analyses are time consuming and their quality depends heavily on the experience of the analyst. Section 3 presents a first attempt at using the Alloy “model finder” to build a tool for automatic fail-secure analysis of early designs. This first trial is not a prototype yet, but rather a draft demonstrating the apparent feasibility of the approach. As stated in the conclusion in Sect. 4, there is much more work to accomplish in this niche, which seems to offer the opportunity for some challenging academic work.

2 Fail-Secure Analysis

In some communities, there is a lot of confusion between the terms “fault-tolerant”, “fail-safe” and “fail-secure”. This confusion is detrimental to attracting attention to the fail-secure design and analysis problems, and impedes understanding and communication. This section tries to clarify the meaning of the term “fail-secure” as used in this paper and other contexts¹, such as the Common Criteria [1].

Figure 1 depicts, in those contexts, the principal meanings of the terms “safety” and “security”, on which are built “fail-safe” and “fail-secure” meanings. It pictures a system whose behavior (functions) influences and is influenced by its data on one side, and interacts with the environment on the other. “Safety” is mainly concerned by the protection of the environment and some of the system’s functionalities. Safety properties ensure that the system’s behavior does no “harm” to its environment or the system itself. “Security” relates mainly to the protection of the (sensitive) data, ensuring that no “villainy” from the environment impacts those data. Adding faults, a system is “fault-tolerant” if its *functional requirements* hold even in the event of faults; and then degrade proportionally to the number of faults (there is no single point of system-wide failure). A system is “fail-safe” if its *safety requirements* hold in the event of faults, and “fail-secure” if its *security requirements* hold in the event of faults. Additionally, all those terms relate to the behavior of the system in the event of faults (and presence of passive attackers if any), but not to its behavior under active attacks. Resistance to active attackers is yet another concept with its own fields of studies. Those distinctions are summarized in Fig. 2.

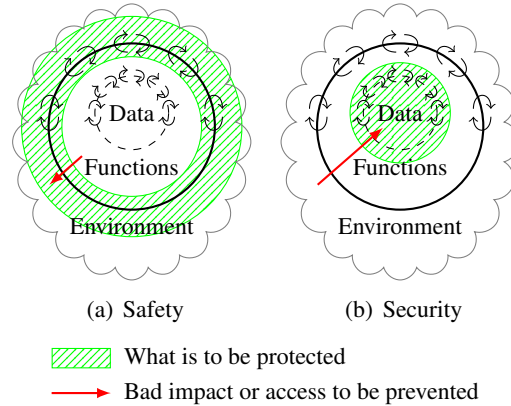


Figure 1: Safety versus security

Analogy with electromagnetic lock. An electromagnetic lock is said to be fail-safe with regard to power failure if it is in the unlocked position when no power is applied to it. While it is fail-secure if it is in the locked position when no power is applied. With the appropriate organizational procedures, assuming that the safety requirements are that employees can always escape and security requirements are that secret documents in safes can not be taken outside, a building can be fail-safe *and* fail-secure with regard to power failure if its office doors are fail-safe and safe doors are fail-secure. In case of power outage, the employees can escape the building and the secret documents are locked inside the safes.

Fail-secure according to the Common Criteria. The Common Criteria are an international standard (ISO/IEC 15408) [1]. This standard is a framework for the security certification of computer information systems. It allows vendors to select functional and assurance requirements that their products meet. Those claims are then verified by independent organizations. One of those functional requirements, FPT_FLS.1 [1, Part 2 (p. 128)] Failure with preservation of secure state, deals with the fail-secure property. It states “that the TSF [(Target Of Evaluation Security Functionality)] preserve a secure state in the face of the identified failures”. The vendor shall state with regard to which failures (power loss, memory error, ...) his product is fully fail-secure; i.e. no security requirement is broken in case of any of those failures. There is no obligation to include FPT_FLS.1. However it appears in many

¹[http://en.wikipedia.org/wiki/\[Electromagnetic_lock|Fail-safe|Life-critical_system\]](http://en.wikipedia.org/wiki/[Electromagnetic_lock|Fail-safe|Life-critical_system])

protection profiles (PP) and security targets (ST), such as the 2011 Trusted Computing Group’s PP for TPM [12] or the 2000 ST of the Naval Research Laboratory’s Network Pump [8].

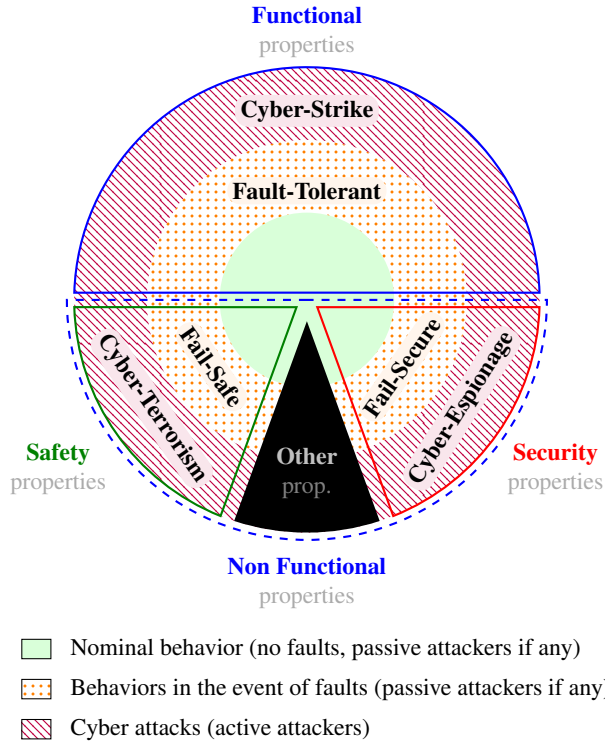


Figure 2: Fault-tolerant, fail-safe and fail-secure

Fidge [10] propose an analysis based on matrices composition. The information flows inside a given atomic component are described in a single matrix (either for one or all operating modes, including different fault behaviors). Those matrices are then “multiplied” following the design of the system to get the overall information flows. This analysis follows the CC approach, specifying the faults taken into account, rather than the NAMILCOM approach which is impervious to the precise type of faults. On the downside, this approach does not scale well. To the exception of this work, there seems to be no work addressing directly the fail-secure design analysis problem. Widening the scope of related works, an information flow analysis taking into account failures could be adapted to provide a fail-secure design analysis for some types of confidentiality and integrity requirements, which are the most common security requirements — excluding access control. However, even if there are lots of information flow analyses [11, 5], to the best of our knowledge, the only information flow analyses handling component failures are those of the SIFA tool [6] and its extension to micro-controller software [7]. SIFA embeds the notion of *faults* into a more generic notion of *mode*. However, SIFA is a circuitry-oriented information flow analysis mainly aimed at evaluating final products. It may therefore not be best suited for a fail-secure analysis of functional and physical architecture designs at the (detailed) specification level.

3 An Alloy-based Feasibility Draft

This section explores the possibility to build a fail-secure (according to NAMILCOM) design analysis tool based on Alloy [4, 3]. As an experiment, Alloy is used to verify if the high level architecture design

Fail-secure according to the NAMILCOM certification. In order to enter the NATO market and deal with sensitive NATO data, security equipments need to get a NAMILCOM certification. Part of this certification process [9] requires a fail-secure analysis of the equipment. The goal of this analysis [2] is not to verify against which failures the equipment is fully fail-secure (not even one security requirement violated), as for the Common Criteria. Rather, it is to verify that the minimum number of faults before sensitive data is potentially compromised (leaked to the outside) is high enough. This number depends on the classification level of the data leaked and involves a notion of functional and physical independence between faults that is not addressed in this paper.

2.1 Related work

To the best of our knowledge, there is really little work on fail-secure design analysis. Rae and

of a fictitious encrypting product is fail-secure against 1 or 2 faults. This product takes as input a key and a message and returns as output the message encrypted by the key. Its security requirement (that is to be preserved even in presence of faults) states that “the product should never directly output the key or message”. The redundant architecture of the product consists in 2 encrypting components followed by a comparator, as seen in Fig. 3. The comparator outputs one of its inputs iff its two inputs are the same.

Alloy is sometimes coined a “model finder” by its authors. It can also be seen as a “meta-model checker”. The Alloy language is used to state constraints that describe a set of models, or structures. It is also used to define properties that are verified with the *Alloy Analyzer*. For a given property (predicate or assertion), this tool finds models, in the provided set, for which this property holds; or states that there is no model, corresponding to the meta-model provided, for which the property holds.

The approach followed by this feasibility draft is to provide a framework that defines a “running” product architecture as a set of components, defined as a function in a given state, connected by connectors (List. 1, lines 1 to 4). Each connector is associated to a symbolic value. The state of a component is either *nominal* or in *failure* (lines 7 to 8). Intrinsically, a component is a function that relates possible output values to input values. For this simplified draft, if a component is in failure state, then its only *functional* constraint (line 5) is that its output values belong to its input values (for each output, the value of one input is directly transferred).

```

2  sig Connector { value: one Value {} }
  abstract sig Function { input: Int -> lone Connector, output: Int -> lone Connector, ... }
    { ... }
4  abstract sig Component extends Function { state: one FunctionalState }
    { state = Failure implies Outputs.value in Inputs.value }
6  one sig Product extends Function {} { ... }
  abstract sig FunctionalState {}
8  one sig Nominal, Failure extends FunctionalState {}
  ...

```

Listing 1: Framework

Instantiating this framework consists in defining components, such as the comparator in List. 2, and describing an architecture based on those components. The remaining of this section is based on an instantiation of this framework for the redundant architecture of the fictitious encrypting product described above. This architecture corresponds to the yellow rounded boxes in Fig. 3. A comparator has two inputs, `input[1]` and `input[2]`, and one output (List. 2, line 10). Its nominal behavior is to return `Null` if `input[1]` and `input[2]` differ, and return the value of `input[1]` otherwise (lines 11–12). “Encryptors” are similarly defined to return the encryption of `input[2]` with the key in `input[1]`. The fictitious product is an encryptor with a redundant architecture (line 15): it has 2 inputs and 1 output (line 16); it is composed of 2 encryptors and 1 comparator (line 17); it shares its inputs with the 2 encryptors `enc1` and `enc2` (lines 19–20); the outputs of `enc1` and `enc2` are internally connected to the inputs of the comparator `cmp` (line 21); and it shares its output with `cmp` (line 22).

```

10 sig Comparator extends Component {} { (input.Connector = 1 + 2) && (output.Connector = 1)
    state in Nominal implies ( equalValues[input[1].value, input[2].value]
12     implies (output[1].value = input[1].value) else output[1].value in Null )
    }
14  ...
  pred Archi_RedundantEnc {
16    Product.input.Connector = 1+2 && Product.output.Connector = 1
    #Encryptor = 2 && one Comparator && Component = Encryptor + Comparator
18    some disj enc1,enc2:Encryptor, cmp:Comparator |

```

```

20 Product.input[1] = enc1.input[1] && Product.input[1] = enc2.input[1] &&
    Product.input[2] = enc1.input[2] && Product.input[2] = enc2.input[2] &&
22 cmp.input[1] = enc1.output[1] && cmp.input[2] = enc2.output[1] &&
    Product.output[1] = cmp.output[1]
    }

```

Listing 2: Architecture instantiation

Verifying that this redundant architecture is fail-secure to a given number of component failures consists in checking a relatively simple assertion. The security requirement stated at the beginning of the section is encoded as “no input value is directly output” (List. 3, line 24)). A product is fail-secure up to n faults iff: it is not running securely only if more than n components are in a failure state (line 25).

```

24 pred secure { no v:Product.Outputs.value | v in Product.Inputs.value }
pred failsecureToN [n:Int] { (not secure) implies #(state.Failure) > n }
26 assert redundantFailsecureToOne { (Archi_RedundantEnc and ...) implies failsecureToN[1] }
assert redundantFailsecureToTwo { (Archi_RedundantEnc and ...) implies failsecureToN[2] }

```

Listing 3: Fail-secure analysis

The redundant architecture of the fictitious product (corresponding to the yellow rounded boxes in Fig. 3) is fail-secure to 1 fault. If a single component is in failure state, whatever the fault, the overall

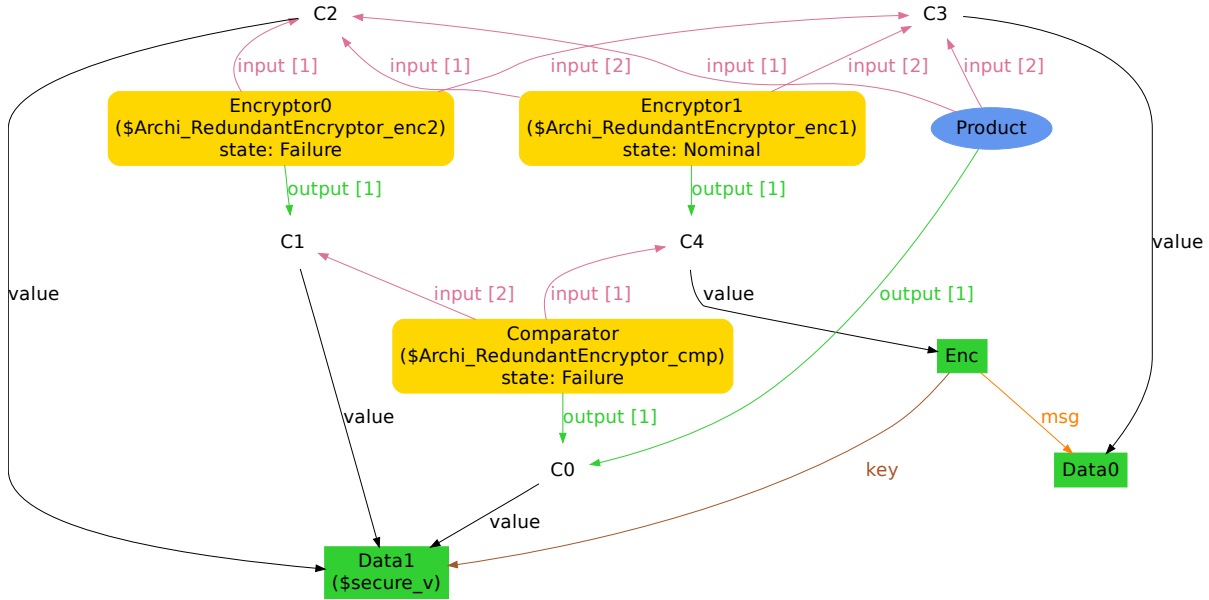


Figure 3: Result of the fail secure analysis against 2 faults for the redundant architecture

product can not leak its inputs. If it is one of the two encryptions which is in failure, the comparator receives two different values as input and returns Null. If it is the comparator which is in failure, then the two encryptions are not faulty and provide the same encrypted values to the comparator which can only output one of its inputs. As expected, Alloy is unable to find a counter-example when asked to verify the assertion corresponding to the redundant architecture being fail-secure up to 1 fault (line 26). However, this redundant architecture is not fail-secure to 2 faults. Figure 3 shows a counter-example returned by Alloy when asked to verify the assertion corresponding to the redundant architecture being

fail-secure up to 2 faults (line 27). In this counter-example, there is a fault in the comparator which, as a result, returns the erroneous value returned by the first encryptor (which is also in failure state and returns directly the encryption key).

4 Conclusion

This framework demonstrates the feasibility to encode into Alloy the main aspects of a fail-secure design analysis. Of course, there is much more work to do for this framework to be usable. First, this framework should integrate the notion of functional and physical independence of faults, and refine the definition of “secure” to reflect the dependence to the classification of data. In a longer run, the Alloy model should be automatically extracted from models expressed in more standard languages, such as UML or SysML.

The development of an automated tool for fail-secure design analysis is a niche market. However, it would be highly beneficial to architects having to do a fail-secure analysis, which is a requirement for dealing with sensitive NATO data. Due to the challenges that need to be resolved and the lack of contenders, the field of fail-secure analysis is an interesting application area for academic work.

References

- [1] Common Criteria Organisation (2012): *Common Criteria for Information Technology Security Evaluation*. ISO/IEC international standard 15408, ISO/IEC. Version 3.1, Revision 4.
- [2] DGA Maîtrise de l’Information (2012): *Guide – Analyse de la Résistance aux Pannes d’un Équipement Cryptographique*. Rapport Technique 2012/066099/DGA.MI/SSI/IPS/AC-DR. Version 4.0. Restricted access.
- [3] Daniel Jackson (2002): *Alloy: A Lightweight Object Modelling Notation*. *ACM Trans. Softw. Eng. Methodol.* 11(2), pp. 256–290, doi:10.1145/505145.505149.
- [4] Daniel Jackson (2012): *Software Abstraction - Logic, Language, and Analysis*. The MIT Press. Available at <http://mitpress.mit.edu/books/software-abstractions>.
- [5] Gurvan Le Guernic (2007): *Confidentiality Enforcement Using Dynamic Information Flow Analyses*. Ph.D. thesis, Kansas State University. Chap.2 Bibliography.
- [6] Tim McComb & Luke Wildman (2005): *SIFA: A Tool for Evaluation of High-grade Security Devices*. In: *Proc. Australasian Conf. Information Security and Privacy, Lecture Notes in Computer Science 3574*, Springer-Verlag, pp. 230–241, doi:10.1007/11506157_20.
- [7] Chris Mills, Colin J. Fidge & Diane Corney (2012): *Tool-Supported Dataflow Analysis of a Security-Critical Embedded Device*. In: *Proc. Australasian Information Security Conf., Conferences in Research and Practice in Information Technology 125*, Australian Computer Society, Inc., pp. 59–70.
- [8] Andrew P. Moore (2000): *Network Pump (NP) Security Target*. CC Security Target NRL/MR/5540–00-8459, Naval Research Laboratory.
- [9] NATO Consultation, Command and Control Board (2009): *INFOSEC Technical and Implementation Directive on Cryptographic Security and Cryptographic Mechanisms*. Directive AC/322-D/0047-REV2 (INV). Restricted access.
- [10] Andrew Rae & Colin J. Fidge (2005): *Information Flow Analysis for Fail-Secure Devices*. *The Computer Journal* 48(1), pp. 17–26, doi:10.1093/comjnl/bxh056.
- [11] Andrei Sabelfeld & Andrew C. Myers (2003): *Language-Based Information-Flow Security*. *IEEE J. on Selected Areas in Communications* 21(1), pp. 5–19, doi:10.1109/JSAC.2002.806121.
- [12] Trusted Computing Group (2011): *Protection profile PC Client Specific Trusted Platform Module Family 1.2; Level 2; Revision 116 (PP TPM F1.2L2)*. CC Protection Profile. Version 1.2.